IN THE SPECIFICATION

Please replace the paragraph beginning on page 39, line 21, with the following:

Algorithm process_HDL_file2() is an augmentation to process_HDL_file() of HDL compiler **342** in order to support the creation of instrumented simulation models. The algorithm is invoked with the name of the top level design entity passed through parameter file and a flag indicating whether the entity being processed is a design entity or an instrumentation entity passed through parameter design_flag (design_flag = TRUE for design entities and FALSE for instrumentation entities). Algorithm process_HDL_file2() (line 5) first checks, by means of routine proto_loaded() (line 15), if the proto for the current entity is already present in memory **44**. If so, processing passes to line [[**105**]] 105. Otherwise, control is passed to line 20 and [[**25**]] 25 where disk **33** of computer system **10** is examined to determine if proto files for the entity and its descendants (including instrumentation entities, if any) exist and are consistent. If so, the appropriate proto files are loaded from disk **10** by routine load_proto() (line 25) creating proto data structures, as necessary, in memory **44** for the current entity and the current entity's descendants including instrumentation entities.

Please replace the paragraph beginning on page 40, line 1, with the following:

If the proto file is unavailable or inconsistent, control passes to line 35 where the current entity HDL file is parsed. For any entities instantiated within the current entity, lines [[**40**]] 40 to [[**55**]] 55 recursively call process_HDL_file2() (line 5) in order to process these descendants of the current entity. Control then passes to line 55 where the design_flag parameter is examined to determine if the current entity being processed is a design entity or an instrumentation entity. If the current entity is an instrumentation entity, control passes to line 80. Otherwise, the current entity is a design entity and lines [[**60**]] 60 to [[**70**]] 70 recursively call process_HDL_file2() (line 5) to process any instrumentation entities instantiated by means of instrumentation instantiation comments. It should be noted that algorithm process_HDL_file2() (line 5) does not allow for instrumentation entities to monitor instrumentation entities. Any instrumentation entity instantiation comments within an instrumentation entity are ignored. Control then passes to line 80 where proto data structures are created in memory **44** as needed for the current entity and any

instrumentation entities. Control then passes to line 90 where the newly created proto data structures are written, as needed to disk **33** of computer system **10**.

Please replace the paragraph beginning on page 40, line 17, with the following:

Control finally passes to line [[**105**]] 105 and [[**110**]] 110 where, if the current entity is a design entity, instance data structures are created as needed for the current entity and the current entity's descendants. If the current entity is an instrumentation entity, routine create_instance() (line [[**110**]] 110) is not called.

Please replace the paragraph beginning on page 73, line 28, with the following:

Signals in *rhs* connectivity expressions can also be connections to signals within entities instantiated within the target design entity. In such a circumstance, the instance names of the entity or entities in the hierarchy enclosing the desired signal are placed before the signal name in hierarchy order, delineated by period (".") characters. For example, the signal in statement **1230** ("Y.P") represents a signal ~~**1204**~~ within design entity **1201**. Signals at any level of the target design hierarchy are thus accessible to instrumentation logic generated by the instrumentation language comments.

Please replace the paragraph beginning on page 74, line 19, with the following:

Statement **1234** utilizes intermediate signal *Q* along with signal ~~**1206**~~ *B* to produce fail event **1241**. The syntax for fail event declaration includes a field denoting the type of event ("fail"), a field giving the event name for the fail event ("failname0"), and a final field denoting the message to associate with the fail. Finally, statement **1236** produces harvest event **1242**.